



IT-Security (ITS) B1

DIKU, E2022



Today's agenda

Part 1: Software vulnerabilities

Part 2: How to find and exploit vulnerabilities (guest)

Lecture plan

Week	Date	Time	Instructor	Topic
36	05 Sep	10-12		Security concepts and principles
	09 Sep	10-12		Cryptographic building blocks
37	12 Sep	10-12	TL	Key establishment and certificate management
	16 Sep	10-12	CJ	User authentication, IAM
38	19 Sep	10-12	CJ	Operating systems security, web, browser and mail security
	23 Sep	10-12	CJ	IT security management and risk assessment
39	26 Sep	10-12	TL	Software security - exploits and privilege escalation
	30 Sep	10-12	TL	Malicious software
40	03 Oct	10-12	CJ	Firewalls and tunnels, security architecture
	07 Oct	10-12	CJ	Cloud and IoT security
41	10 Oct	10-12	TL	Intrusion detection and network attacks
	14 Oct	10-12	TL	Forensics
42				Fall Vacation - No lectures
43	24 Oct	10-12	CJ	Privacy and GDPR
	28 Oct	10-12	CJ	Privacy engineering
44	31 Oct	10-11	Guest	Special topic
		11-12	TL,CJ	Exam Q/A

<https://github.com/diku-its/its-e2022/blob/main/lectureplan2022.md>



Vulnerabilities



Vulnerabilities defined

Software contains **bugs**

A **vulnerability** is a bug that can be exploited by an attacker

An **exploit** is a piece of code that takes advantage of a vulnerability

Vulnerabilities are exploited to run **malware**

(Not all bugs can be exploited)

(Not all vulnerabilities matter the same / are equally risky)

There are many kinds of vulnerabilities

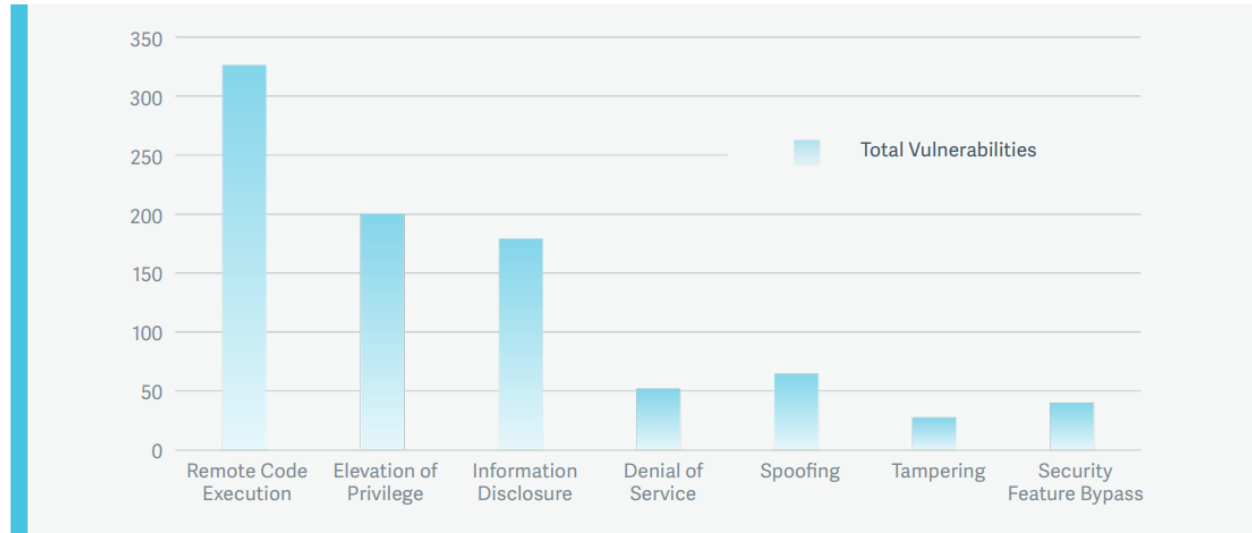


Figure 1: Breakdown of Microsoft Vulnerability Categories (2019)

Vulnerabilities' role in an attack

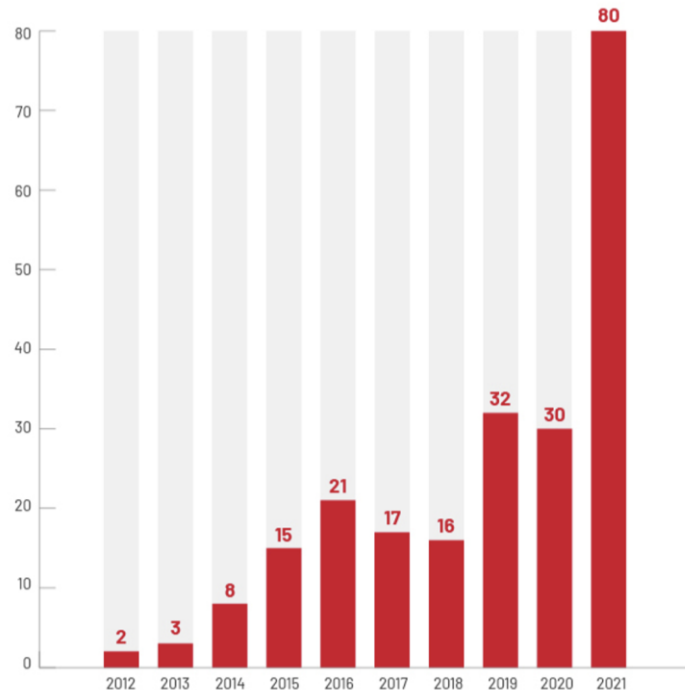
ATT&CK Matrix for Enterprise

layouts show sub-techniques hide sub-techniques

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Command and Control	Exfiltration	Impact
9 techniques	10 techniques	18 techniques	12 techniques	34 techniques	14 techniques	24 techniques	9 techniques	16 techniques	16 techniques	9 techniques	13 techniques
Drive-by Compromise	Command and Scripting Interpreter	Account Manipulation	Abuse Elevation Control Mechanism	Abuse Elevation Control Mechanism	Brute Force	Account Discovery	Exploitation of Remote Services	Archive Collected Data	Application Layer Protocol	Automated Exfiltration	Account Access Removal
Exploit Public-Facing Application	Exploitation for Client Execution	BITS Jobs	Access Token Manipulation	Access Token Manipulation	Credentials from Password Stores	Application Window Discovery	Internal Spearphishing	Audio Capture	Communication Through Removable Media	Data Transfer Size Limits	Data Destruction
External Remote Services	Inter-Process Communication	Boot or Logon Autostart Execution	Boot or Logon Autostart Execution	BITS Jobs	Exploitation for Credential Access	Browser Bookmark Discovery	Lateral Tool Transfer	Automated Collection	Data Encoding	Exfiltration Over Alternative Protocol	Data Encrypted for Impact
Hardware Additions	Native API	Boot or Logon Initialization Scripts	Boot or Logon Initialization Scripts	Deobfuscate/Decode Files or Information	Forced Authentication	Cloud Service Dashboard	Remote Service Session Hijacking	Clipboard Data	Data Obfuscation	Exfiltration Over C2 Channel	Data Manipulation
Phishing	Scheduled Task/Job	Browser Extensions	Boot or Logon Initialization Scripts	Direct Volume Access	Input Capture	Cloud Service Discovery	Remote Services	Data from Cloud Storage Object	Dynamic Resolution	Exfiltration Over Other Network Medium	Defacement
Replication Through Removable Media	Shared Modules	Compromise Client Software Binary	Create or Modify System Process	Execution Guardrails	Man-in-the-Middle	Domain Trust Discovery	Replication Through Removable Media	Data from Information Repositories	Encrypted Channel	Exfiltration Over Physical Medium	Disk Wipe
Supply Chain Compromise	Software Deployment Tools	Event Triggered Execution	Event Triggered Execution	Exploitation for Defense Evasion	Modify Authentication Process	File and Directory Permissions Modification	Software Deployment Tools	Data from Local System	Fallback Channels	Firmware Corruption	Endpoint Denial of Service
Trusted Relationship	System Services	Create Account	Group Policy Modification	File and Directory Permissions Modification	Network Sniffing	Network Service Scanning	Taint Shared Content	Use Alternate Authentication Material	Ingress Tool Transfer	Inhibit System Recovery	Resource Hijacking
Valid Accounts	User Execution	Create or Modify System Process	Hide Artifacts	Group Policy Modification	OS Credential Dumping	Network Sniffing	Use Alternate Authentication Material	Data from Removable Media	Multi-Stage Channels	Network Denial of Service	Service Stop
	Windows Management Instrumentation	Event Triggered Execution	Hijack Execution Flow	Impair Defenses	Hijack Execution Flow	Permission Groups Discovery	Peripheral Device Discovery	Data Staged	Non-Application Layer Protocol	Scheduled Transfer	System Shutdown/Reboot
		External Remote Services	Indicator Removal on Host	Impair Defenses	Indicator Removal on Host	Process Discovery	Query Registry	Email Collection	Non-Standard Port	Transfer Data to Cloud Account	
		Hijack Execution Flow	Process Injection	Indirect Command Execution	Indirect Command Execution	Software Discovery	Remote System Discovery	Input Capture	Protocol Tunneling		
		Implant Container Image	Scheduled Task/Job	Masquerading	Masquerading	System Information Discovery	Screen Capture	Man in the Browser	Proxy		
		Office Application Startup	Valid Accounts	Modify Authentication Process	Modify Authentication Process	System Network Configuration Discovery	Video Capture	Man-in-the-Middle	Remote Access Software		
		Pre-OS Boot		Modify Cloud Compute Infrastructure	Modify Cloud Compute Infrastructure	System Network Connections Discovery		Traffic Signaling	Web Service		
		Scheduled Task/Job		Modify Registry	Modify Registry						
		Server Software Component		Obfuscated Files or Information	Obfuscated Files or Information						

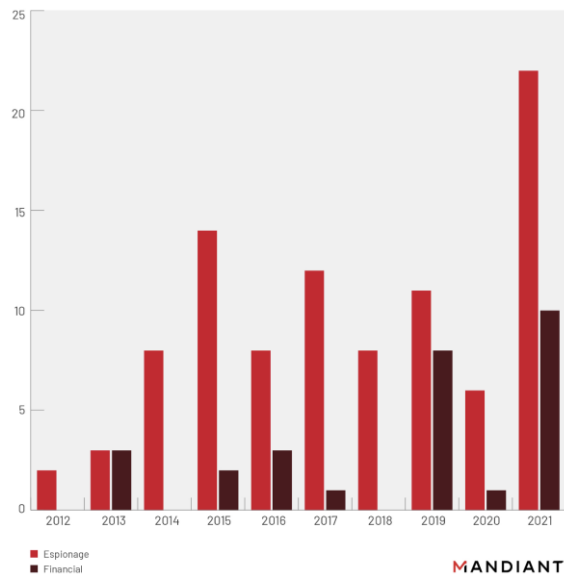
Zero-day vulnerabilities

A zero-day vulnerability is a vulnerability that defenders have previously been unaware of, and for which they have had zero days to produce a fix or workaround, providing attackers the best opportunity to attack affected systems.

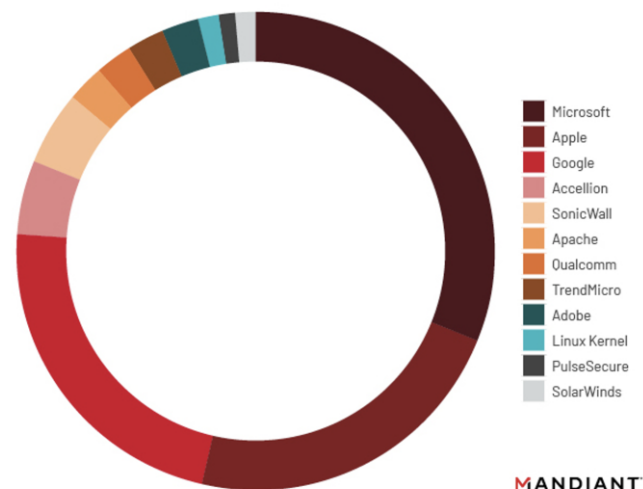


Mandiant on zero-days exploited in the wild

Espionage Actors Lead Growth in Zero-Day Exploitation



Vendors Targeted by Zero-Day Exploits





Known Exploited Vulnerabilities (KEV) catalog

The Cybersecurity and Infrastructure Security Agency (CISA) is an agency responsible for strengthening cybersecurity across the US government

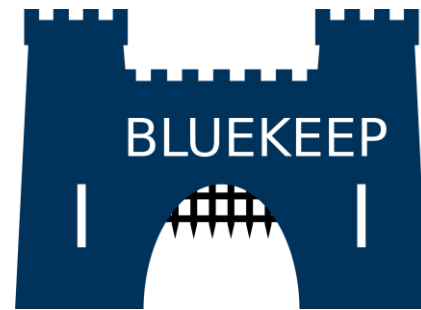
CISA maintains an authoritative source of vulnerabilities that have been exploited in the wild: the Known Exploited Vulnerability (KEV) catalog

cveID	vendorProject	dateAdded	dueDate
CVE-2022-40139	Trend Micro	15-09-2022	06-10-2022
CVE-2013-6282	Linux	15-09-2022	06-10-2022
CVE-2013-2597	Code Aurora	15-09-2022	06-10-2022
CVE-2013-2596	Linux	15-09-2022	06-10-2022
CVE-2013-2094	Linux	15-09-2022	06-10-2022
CVE-2010-2568	Microsoft	15-09-2022	06-10-2022
CVE-2022-37969	Microsoft	14-09-2022	05-10-2022
CVE-2022-32917	Apple	14-09-2022	05-10-2022
CVE-2022-3075	Google	08-09-2022	29-09-2022
CVE-2022-28958	D-Link	08-09-2022	29-09-2022
CVE-2022-27593	QNAP	08-09-2022	29-09-2022
CVE-2022-26258	D-Link	08-09-2022	29-09-2022
CVE-2020-9934	Apple	08-09-2022	29-09-2022
CVE-2018-7445	MikroTik	08-09-2022	29-09-2022
CVE-2018-6530	D-Link	08-09-2022	29-09-2022
CVE-2018-2628	Oracle	08-09-2022	29-09-2022
CVE-2018-13374	Fortinet	08-09-2022	29-09-2022
CVE-2017-5521	NETGEAR	08-09-2022	29-09-2022
CVE-2011-4723	D-Link	08-09-2022	29-09-2022

This is a vulnerability

BlueKeep (CVE-2019-0708) is a vulnerability that was discovered in Microsoft's Remote Desktop Protocol (RDP) implementation, which allows for the possibility of remote code execution.

First reported in May 2019, Microsoft issued a security patch (including an out-of-band update for several versions of Windows that have reached their end-of-life, such as Windows XP) on 14 May 2019. On 6 September 2019, a Metasploit exploit of the wormable BlueKeep security vulnerability was publicly released.



Kevin Beaumont ✓
@GossiTheDog

CVE-2019-0708 RDP vulnerability megathread, aka BlueKeep.

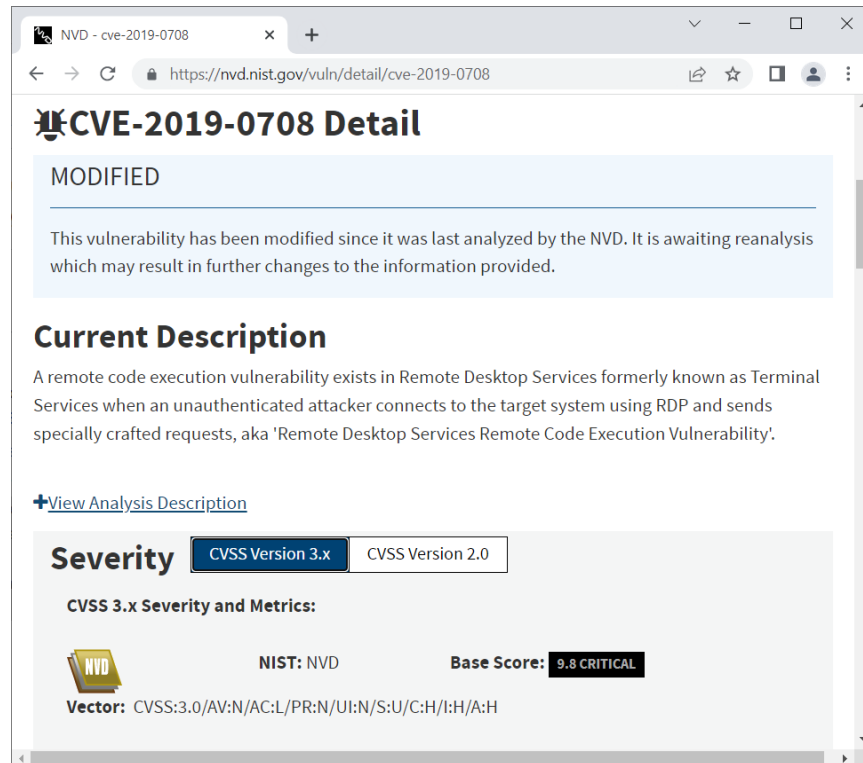
Going to nickname it BlueKeep as it's about as secure as the Red Keep in Game of Thrones, and often leads to a blue screen of death when exploited.

12.47 AM · 15. maj 2019 · Twitter for iPhone

This is a vulnerability

BlueKeep (CVE-2019-0708) is a vulnerability that was discovered in Microsoft's Remote Desktop Protocol (RDP) implementation, which allows for the possibility of remote code execution.

First reported in May 2019, Microsoft issued a security patch (including an out-of-band update for several versions of Windows that have reached their end-of-life, such as Windows XP) on 14 May 2019. On 6 September 2019, a Metasploit exploit of the wormable BlueKeep security vulnerability was publicly released.



The screenshot shows the NVD (National Vulnerability Database) detail page for CVE-2019-0708. The browser address bar shows the URL <https://nvd.nist.gov/vuln/detail/cve-2019-0708>. The page title is "CVE-2019-0708 Detail". A blue box labeled "MODIFIED" contains the text: "This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting reanalysis which may result in further changes to the information provided." Below this is the "Current Description" section, which states: "A remote code execution vulnerability exists in Remote Desktop Services formerly known as Terminal Services when an unauthenticated attacker connects to the target system using RDP and sends specially crafted requests, aka 'Remote Desktop Services Remote Code Execution Vulnerability'." There is a link to "View Analysis Description". The "Severity" section shows two boxes: "CVSS Version 3.x" and "CVSS Version 2.0". Below this, the "CVSS 3.x Severity and Metrics:" section displays a "NIST: NVD" logo, a "Base Score: 9.8 CRITICAL" badge, and the "Vector: CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H".

NVD - cve-2019-0708

https://nvd.nist.gov/vuln/detail/cve-2019-0708

CVE-2019-0708 Detail

MODIFIED

This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting reanalysis which may result in further changes to the information provided.

Current Description


A remote code execution vulnerability exists in Remote Desktop Services formerly known as Terminal Services when an unauthenticated attacker connects to the target system using RDP and sends specially crafted requests, aka 'Remote Desktop Services Remote Code Execution Vulnerability'.

[+View Analysis Description](#)

Severity

CVSS Version 3.x CVSS Version 2.0

CVSS 3.x Severity and Metrics:

 **NIST: NVD** **Base Score: 9.8 CRITICAL**

Vector: CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H



CVE and CVSS

The Common Vulnerabilities and Exposures (CVE) system provides a reference-method for publicly known vulnerabilities.

The Common Vulnerability Scoring System (CVSS) is a free and open industry standard for assessing the severity of vulnerabilities.

CVSS v3.0 Severity and Metrics:

Base Score: 9.8 CRITICAL

Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

Impact Score: 5.9

Exploitability Score: 3.9

Attack Vector (AV): Network

Attack Complexity (AC): Low

Privileges Required (PR): None

User Interaction (UI): None

Scope (S): Unchanged

Confidentiality (C): High

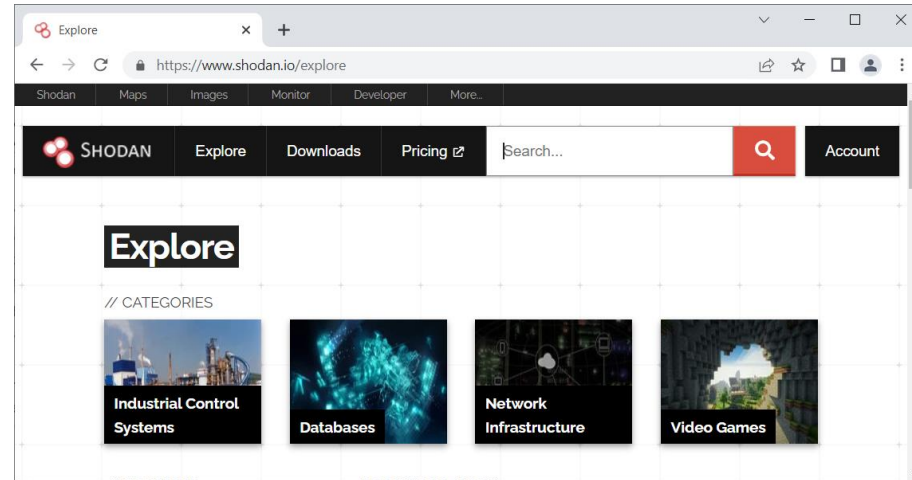
Integrity (I): High

Availability (A): High

Shodan and BleeKeep

Shodan is a search engine that lets users search for various types of servers (webcams, routers, servers, etc.) connected to the internet using a variety of filters.

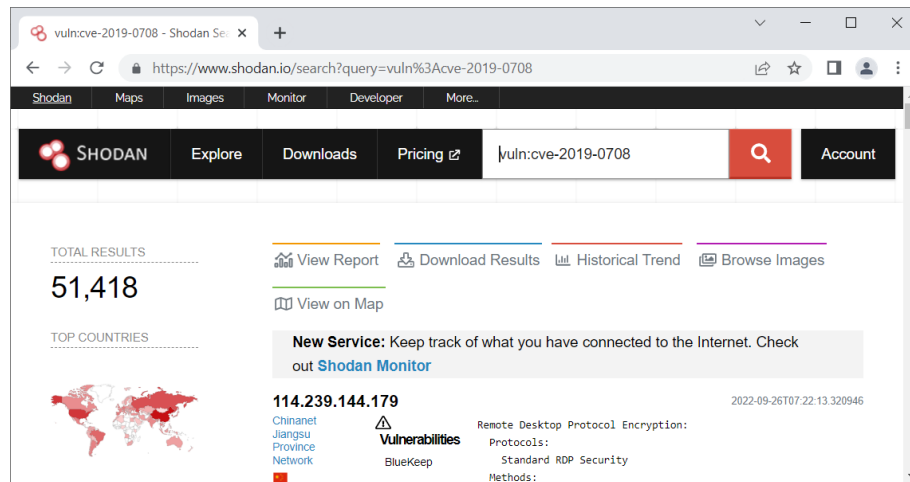
This can be information about the server software, what options the service supports, a welcome message or anything else that the client can find out before interacting with the server.



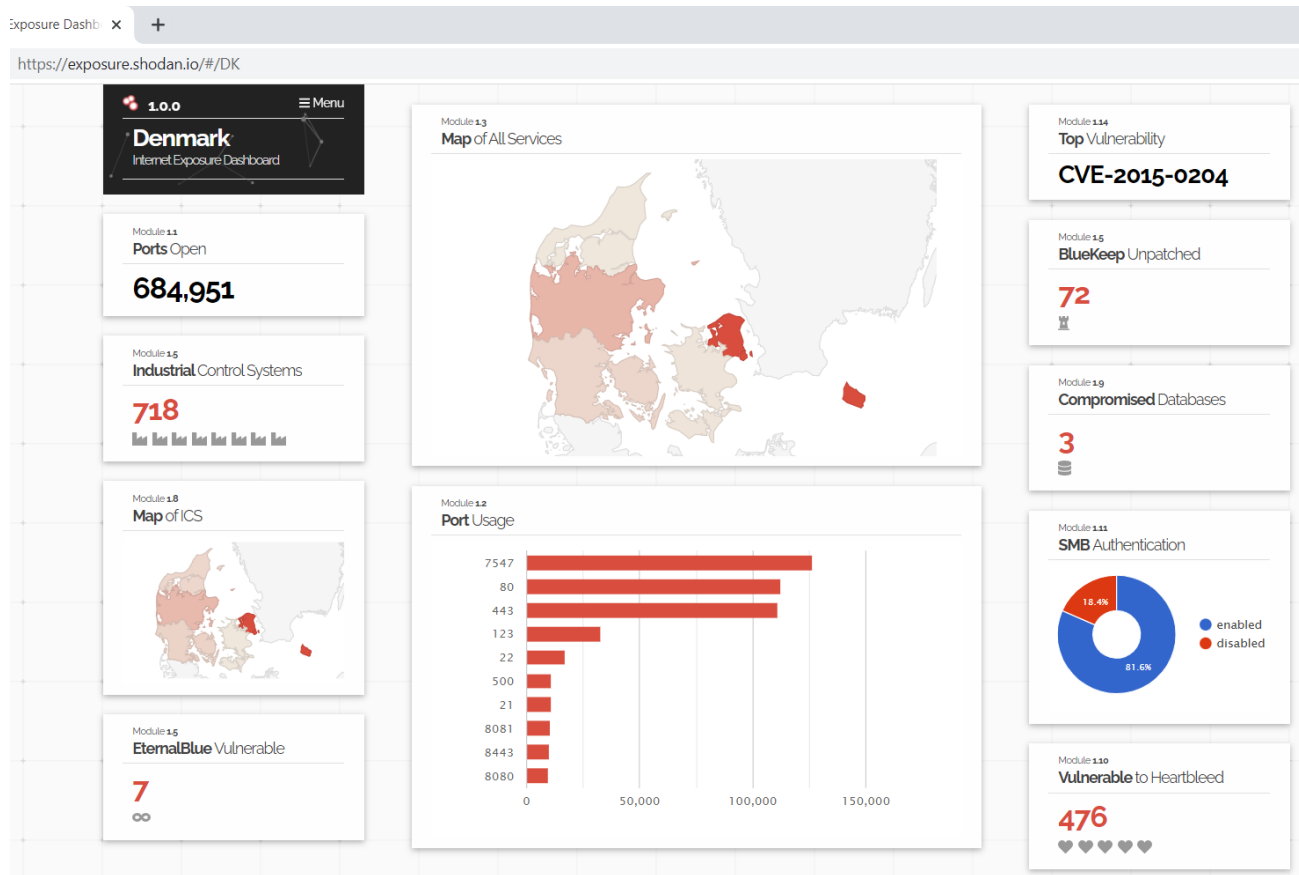
Shodan and BleeKeep

Shodan is a search engine that lets users search for various types of servers (webcams, routers, servers, etc.) connected to the internet using a variety of filters.

This can be information about the server software, what options the service supports, a welcome message or anything else that the client can find out before interacting with the server.



Shodan and “Denmark”





Wrap-up

Lecture plan

Week	Date	Time	Instructor	Topic
36	05 Sep	10-12		Security concepts and principles
	09 Sep	10-12		Cryptographic building blocks
37	12 Sep	10-12		Key establishment and certificate management
	16 Sep	10-12		User authentication, IAM
38	19 Sep	10-12		Operating systems security, web, browser and mail security
	23 Sep	10-12		IT security management and risk assessment
39	26 Sep	10-12		Software security - exploits and privilege escalation
	30 Sep	10-12		Malicious software
40	03 Oct	10-12		Firewalls and tunnels, security architecture
	07 Oct	10-12		Cloud and IoT security
41	10 Oct	10-12		Intrusion detection and network attacks
	14 Oct	10-12		Forensics
42				Fall Vacation - No lectures
43	24 Oct	10-12		Privacy and GDPR
	28 Oct	10-12		Privacy engineering
44	31 Oct	10-11		Special topic
		11-12		Exam Q/A

<https://github.com/diku-its/its-e2022/blob/main/lectureplan2022.md>



IT-Security (ITS) B1

DIKU, E2022



Types of vulnerabilities, include:

Format string

Overflow

Over-read

Load order

Use-after-free

Dangling pointers

Code injection

Command injection

Race conditions

Typos, and more



Where's the bug?



```
#include <stdio.h>
```

```
int main () {  
    int i;  
    printf("Enter a value: ");  
    scanf("%d", &i);
```

```
    if (i < 0)  
        goto fail;  
    if (i > 100)  
        goto fail;  
    goto fail;  
    if (i%2 == 0)  
        goto fail;
```

```
    return;
```

```
fail:  
    printf("Fail\n");  
    return;  
}
```

```
$ ./a.out  
Enter a value: 2  
Fail
```

```
$ ./a.out  
Enter a value: 3  
Fail
```



```
#include <stdio.h>
```

```
int main () {  
    int i;  
    printf("Enter a value: ");  
    scanf("%d", &i);
```

```
    if (i < 0)  
        goto fail;  
    if (i > 100)  
        goto fail;  
    //goto fail;  
    if (i%2 == 0)  
        goto fail;
```

```
    return;
```

```
fail:  
    printf("Fail\n");  
    return;  
}
```

```
$ ./a.out  
Enter a value: 2  
Fail
```

```
$ ./a.out  
Enter a value: 3  
Fail
```

Apple iOS Goto Fail

```
1 static OSStatus
2 SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
3                                   uint8_t *signature, UInt16 signatureLen)
4 {
5     OSStatus      err;
6     ...
7
8     if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
9         goto fail;
10    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
11        goto fail;
12    if ((err = SSLHashSHA1.update(&hashCtx, &signature)) != 0)
13        goto fail;
14    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
15        goto fail;
16    ...
17 fail:
18    SSLFreeBuffer(&signedHashes);
19    SSLFreeBuffer(&hashCtx);
20    return err;
21 }
```




```
#include <stdio.h>
#include <string.h>

int main () {

    char buf[20] = "http://www.diku.dk";
    char shh[30] = "mumstheword";
    char out[64];
    int chars;

    printf("Buffer contents: %s\n", buf);

    printf("Chars to copy: ");
    scanf("%d", &chars);


    memcpy(out, buf, chars);

    printf("Copied: ");
    fwrite(out, chars, 1, stdout);
    printf("\n");

    return(0);
}
```

```
$ ./a.out
Buffer contents: http://www.diku.dk
Chars to copy: 12
Copied: http://www.d

$ ./a.out
Buffer contents: http://www.diku.dk
Chars to copy: 50
Copied: http://www.diku.dk0LHmumstheword
```



```
#include <stdio.h>
#include <string.h>
```

```
int main () {
```

```
    char buf[20] = "http://www.diku.dk";
    char shh[30] = "mumstheword";
    char out[64];
    int chars;
```

```
    printf("Buffer contents: %s\n", buf);
```

```
    printf("Chars to copy: ");
    scanf("%d", &chars);
```

```
    if (chars > sizeof(buf)) chars = sizeof(buf);
    memcpy(out, buf, chars);
```

```
    printf("Copied: ");
    fwrite(out, chars, 1, stdout);
    printf("\n");
```

```
$ ./a.out
```

```
Buffer contents: http://www.diku.dk
```

```
Chars to copy: 12
```

```
Copied: http://www.d
```

```
$ ./a.out
```

```
Buffer contents: http://www.diku.dk
```

```
Chars to copy: 50
```

```
Copied: http://www.diku.dk0LHmumstheword
```

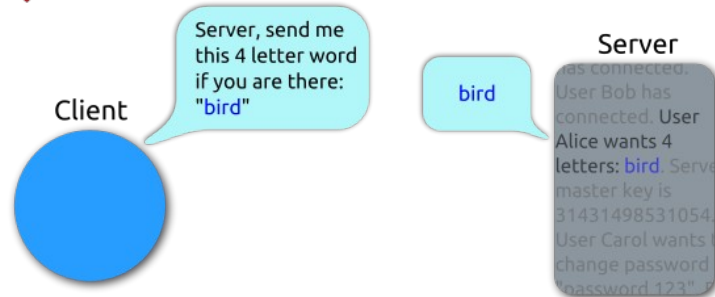
The HeartBleed Bug

Heartbleed was a security bug in the OpenSSL cryptography library, which is a widely used implementation of the Transport Layer Security (TLS) protocol. It was introduced into the software in 2012 and publicly disclosed in April 2014.

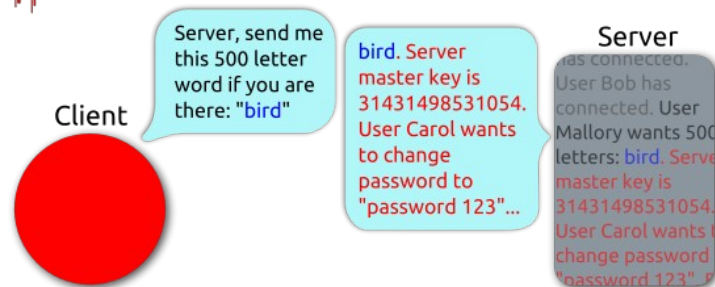
Heartbleed could be exploited regardless of whether the vulnerable OpenSSL instance is running as a TLS server or client. It resulted from improper input validation (due to a missing bounds check) in the implementation of the TLS heartbeat extension.




Heartbeat – Normal usage



Heartbeat – Malicious usage





```
#include <stdio.h>
#include <stdlib.h>
```

```
int main(int argc, char **argv)
{


    printf("Current time: ");
    fflush(stdout);
    system("date");
    return 0;

}
```

```
$ ./a.out
Current time: Mon Sep 26 10:35:47 CEST 2022
```

```
$ export PATH=`pwd`: $PATH
$ echo -e '#!/bin/sh\nnecho "Hello"' > date
$ chmod 700 date
```

```
$ ./a.out
Current time: Hello
```



```
#include <stdio.h>
#include <stdlib.h>
```

```
int main(int argc, char **argv)
{

    printf("Current time: ");
    fflush(stdout);
    system("/bin/date");
    return 0;

}
```

```
$ ./a.out
Current time: Mon Sep 26 10:35:47 CEST 2022
```

```
$ export PATH=`pwd`: $PATH
$ echo -e '#!/bin/sh\nnecho "Hello"' > date
$ chmod 700 date
```

```
$ ./a.out
Current time: Hello
```



Real-world example: PlugX


PlugX drops

- A legitimate NVIDIA file (NvSmart.exe)

- A malicious DLL (NvSmartMax.dll)

Normally, NvSmart.exe would load a legitimate NvSmartMax.dll

But if a (malicious) version the DLL file is located in the same directory, this will load instead



```
#!/usr/bin/perl

open(FH, $ARGV[0]);

while(<FH>)
{
    print $_;
}

close(FH);
```

```
$ ./code.pl code.pl
#!/usr/bin/perl
```

```
open(FH, $ARGV[0]);

while(<FH>)
{
    print $_;
}
```

```
close(FH);
```

```
$ ./code.pl 'ls -l code.pl'|
-rwx----- 1 user user 79 Sep 26 10:41 code.pl
```



```
#!/usr/bin/perl
```

```
open(FH, "< ".$ARGV[0]); #force read open with '<'
```

```
while(<FH>)  
{  
    print $_;  
}
```

```
close(FH);
```

```
$ ./code.pl code.pl  
#!/usr/bin/perl
```

```
open(FH, $ARGV[0]);
```

```
while(<FH>)  
{  
    print $_;  
}
```

```
close(FH);
```


```
$ ./code.pl 'ls -l code.pl'  
-rwx----- 1 user user 79 Sep 26 10:41 code.pl
```




Explanation

According to the Perl documentation

If filename ends with a "|", filename is interpreted as a command which pipes output



```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char **argv)
{
    char buffer[64];
    strncpy(buffer, argv[1], sizeof(buffer));


    printf("You entered: ");
    printf(buffer);
    printf("\n");
}
```

```
$ ./a.out A
You entered: A
```

```
$ ./a.out %s
You entered: You entered:
```

```
$ ./a.out %x
You entered: 510a2000
```

```
$ ./a.out %x%x
You entered: 437a00041569e0
```



```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char **argv)
{
    char buffer[64];
    strncpy(buffer, argv[1], sizeof(buffer));


    printf("You entered: ");
    printf("%s", buffer);
    printf("\n");
}
```

```
$ ./a.out A
You entered: A
```

```
$ ./a.out %s
You entered: You entered:
```

```
$ ./a.out %x
You entered: 510a2000
```

```
$ ./a.out %x%x
You entered: 437a00041569e0
```



```
#include <string.h>
```


```
void foo (char *bar)
{
    char c[12];
    strcpy(c, bar);
}
```

```
int main (int argc, char **argv)
{
    foo(argv[1]);
}
```

```
$ ./6.out A
```

```
$ ./6.out AAAAAAAAAAAAAAA
```

```
$ ./6.out AAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault
```



```
#include <string.h>
```

```
void foo (char *bar)
{
    char c[12];
    strncpy(c, bar, sizeof(c));
}
```

```
int main (int argc, char **argv)
{
    foo(argv[1]);
}
```

```
$ ./6.out A
```

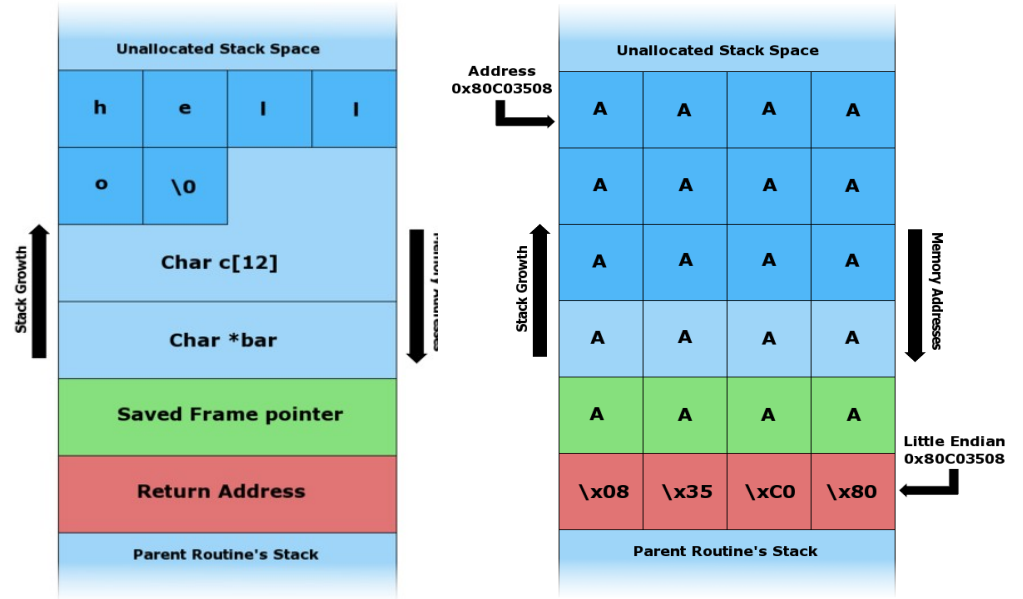
```
$ ./6.out AAAAAAAAAAAAAAA
```

```
$ ./6.out AAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault
```

```
#include <string.h>
```

```
void foo (char *bar)
{
    char c[12];
    strcpy(c, bar);
}
```

```
int main (int argc, char **argv)
{
    foo(argv[1]);
}
```





Some countermeasures

Stack canaries

Check stack not altered when function returns

Data execution prevention (DEP)

Prevent the execution of data on the stack or heap

Address space layout randomization (ASLR)

Rearrange memory positions to make successful exploitation more difficult